



下载APP



21 | 高性能设计，一切都围绕着契约精神

2020-12-11 乔新亮

乔新亮的CTO成长复盘

[进入课程 >](#)



讲述：乔新亮

时长 21:15 大小 19.46M



你好，我是乔新亮。这一讲，我们来聊聊如何实现架构的高性能设计。

前面我们讲过，产品思维有两个核心关键词：“契约精神”和“洞察人性”。其实高性能设计，也和契约精神是密切相关的。我将其总结为：高性能设计，一切围绕着契约精神。

你可能会想，高性能设计不就是可以支撑大流量、高并发的架构设计吗？和契约精神又有什么关系？

其实关系很大。所谓的高性能，实际上，是与业务强相关的。比如，对于一台网络游戏服务器来说，可以支撑 400 名玩家同时在线，就算高性能；对于一台流媒体服务器来说，可以支撑 10,000 用户同时在线观看，就算高性能。



当然，数据可能不准确，但逻辑大体如此。在实际的业务场景中，还会有许多其他技术指标被引入。比如游戏需要关注连接稳定性、视频需要关注延时，等等。

明确了性能和业务的关系，下面我们就来聊聊，如何进行高性能设计，也借此加深对契约精神的认知和理解。

契约精神是高性能架构设计的“拦路虎”

首先，我们要清楚，高性能设计可以大体拆解为两大步骤：

1. 清晰描述、定义性能目标；
2. 设计并实现这个目标。

而高性能架构的设计目标，是通过三类指标来具体定义的，分别是：

1. 系统响应时间，一般指服务 / 交易处理的时间，也包括网络响应时间；
2. 系统吞吐量，一般指系统的每秒交易量，通常需要结合并发量共同描述；
3. 系统容量，通常代表系统的可用资源数量，包括硬盘容量、网络带宽等。

但要注意：虽然我们有三类高性能指标，但彼此不能孤立地看待，否则就会出现问題。比如，在定义系统响应时间目标时，如果不与系统吞吐量关联起来，就会变得缺乏实践意义——一旦流量的压力高峰到来，系统往往就不遂人愿了。

这意味着，我们要学会识别架构设计中的组件（或服务），首先弄清哪些组件需要明确高性能指标，再针对性地用以上三个指标做清晰定义。

除此之外，对于系统响应时间，我们还有一个更直观的监控指标，叫做 Top 百分数（Top Percentile），简称 TP。比如，TP 90 = 2s，意思是，90% 的请求的响应时间都在 2s 内；那么 TP 99 = 2s 的意思就是 99% 的请求的响应时间都在 2s 内；我们一般说的 RT = 2s，指的是平均响应时间，可以作为参考，但实际意义可能并不大。我个人一般会重点看 TP 90 和 TP 99 的数据。

当我们明确了 TP 90 或者 TP 99 这一有关系统响应时间的指标时，一个交付给用户的“契约”就出现了。守好诸多类似的契约，也就是保证了高性能设计。但学习需要适当

地进行延展思考，比如此时，我们应该多问自己一句：这个契约意味着什么？

对于没有经验的程序员或架构师来说，这里其实存在一个隐含的假设，某种意义上，也算是一种侥幸心理：如果该系统在业务流量低峰场景下，满足了响应时间等性能指标，也就意味着，该系统在任何场景下，都能满足这些指标。

你可能会觉得有点绕，那我换个说法来描述刚刚签订的高性能“契约”：本系统、组件、服务承诺在任何情况、任何资源、任何压力高峰下，都保证 $TP_{90} = 2s$ 。

是不是立刻就有了不一样的感觉？好像契约发生了变化，光是读出来，就让人心里一抖。

其实到此时，还谈不上契约是否发生了变化，因为打从一开始，这个契约就是不明确、不清晰、不具体的。

注意，这里还没有谈架构设计和研发能力，问题就已经出现了。很多时候，我们会被工作中出现的一些实际问题所困扰，其根源往往不在于相关责任人缺乏设计能力，而在于签订的“契约”并不明确。团队不可能实现和保卫一个“空中花园”。

所以，我总说，高性能设计，一切围绕着契约精神。所以接下来，我们的内容都将围绕着“如何将契约描述清楚”这件事来展开。

有上限的“契约”才能交付

我曾经培养过许多架构师。一开始在做高性能设计时，他们常常会这样描述给用户的“契约”：

“该架构最高支持 200 万并发流量， $TP_{90} = 2s$ 。”

在不考虑业务需求的情况下，这个契约看似是没问题的。这意味着：对于第 1 名连接服务器的用户来说， $TP_{90} = 2s$ ；但对于第 300 万名连接服务器的用户来说，契约又是什么呢？答案是，没有，或者说比较模糊。

如果目标不清晰，隐患就埋下了。高性能设计非常看重系统响应时间的一致性，尤其是在不同的服务阶段，并发量和吞吐量发生变化的时候。

对于管理者或架构师而言，理想的契约应该是“保证设计流量内的用户 $TP\ 90 = 2s$ ，超出并发限制的用户，则暂时不在契约承诺的范围内。”。

这就要求我们通过四大步骤去思考所谓高性能架构的设计问题：

第一步，当服务器不超过 200 万并发用户时， $TP\ 90 = 2s$ 。

第二步，当连接服务器的并发用户数量超过 200 万，达到 250 万时，保证有 200 万用户的 $TP\ 90 = 2s$ ，拒绝其他用户的连接请求。

第三步，为设计容量外的用户提供服务器排队机制，并附带具体、明确的系统提示。

第四步，3 分钟内完成扩容，保证并发用户数量小于等于 250 万时，任何在线用户的 $TP\ 90 = 2s$ 。

其实，这样才构成了一个可以履行的契约：无论流量波动如何，该架构始终支持 200 万用户的 $TP\ 90 = 2s$ ；高峰时期，其他用户会被拒绝访问或进行排队，但系统能在三分钟内完成扩容，支持 250 万并发在线用户，并且保证其中任何一个用户的 $TP\ 90 = 2s$ 。

在某种程度上，这也是对业务诉求的精确表达，对于业务发展来说至关重要。很多同学设计的架构和契约，压根没有控制上限。他们想当然地认为，流量不可能出现太大的变化或波动。对于用户来说，这无异于一张“空头支票”。

所以说，要做高性能设计，一定要明确目标，并交付给用户；没有清晰的目标，也就没有针对性的设计，怎么能实现高性能的架构呢？

目标清晰后，我们再来看看，高性能架构如何设计并落地。

实现高性能架构的关键技术点

落地也不难，有三项工作最为关键，分别是：

1. 为架构做好“保护系统”；
2. 使架构具备扩容能力；

3. 提升系统各组件处理能力。

保护系统，是为应对容量规划外的过载而设计的，通过流量控制来具体实现。

所谓流量控制，就是当实际并发压力，超过设计性能的时候，人为阻断服务器连接，告知用户需要排队或“稍后再试”。

经常玩游戏的同学可能知道，对于网络游戏来说，流量控制是个基础设计，基本上所有的服务器都有“排队机制”。

流量控制有两种具体的实践，一种是面向连接数做控制，一种是面向用户数做控制。前者让用户在不断尝试连接时，有一定成功的可能；后者则保证用户对系统的期望是一致的——要么可以登录、要么不能登录。具体应该选择哪种方式，取决于业务的实际诉求。

而对于扩容能力来说，一般要包括储备额外的计算资源和具备快速弹性扩容能力。

你可能会说，既然已经有计算资源了，为什么不提前进行扩容呢？

工作经验比较丰富的同学可能会知道其中的“隐情”：一般来说，该做的系统扩容，通常很早就完成了，架构设计真正缺乏的是应急手段。毕竟，你不能“不分青红皂白”，将每个系统的应急扩容都提前做好。一个经营稳健的企业，是不会允许这样的开销出现的。

另一个关键要素在于扩容速度。扩容工作是需要一小时、一分钟，还是一秒钟完成，其差别非常之大。“天下武功，唯快不破”，就算架构设计做得不好，如果响应够快，很多情况下，也是能解决问题的。

无论是利用公有云扩容，还是私有云扩容，目标都应该是一致的。想象一下这样的场景：

双十一来了，你通过监控平台，看到在线用户数量快速增长，很快突破了 200 万，流量控制机制开始生效，新用户进入排队序列。你考虑了 3 秒钟，然后做了一个决定：系统需要扩容。于是，你轻轻地输入一个数字，按下回车。3 秒钟后，排队序列消失了。你对旁边的同学说，继续监控吧！然后悠闲地端起茶杯，离开了座位。

这将是一种多么美妙的工作状态？更棒的是，以上场景完全有可能实现。

至此，我们聊过了契约，也聊过了应急处理，也就意味着，我们把握住了高性能设计的“头”和“尾”。最后，我们再简单谈一下“中间”部分，也就是对系统各组件处理能力的提升。

在高性能设计中，对于每个组件 / 服务都要确定目标，进行设计，然后进行评审和测试，确保满足性能需求。对于架构负责人来说，性能设计一定要尽早开始，具体工作内容包括：

需求早期收集

容量分析

估算与建模

技术研究

设计、开发、跟踪

测试计划执行

风险与绩效管理

实时监控与容量管理

工作内容和流程可能有些多，但这更多属于标准化的研发管理流程，这里列出来仅供参考，实际执行的时候，需要参考具体的业务和企业情况做调整。

在这中间，有一点需要额外注意，我们称之为“对系统资源的争抢问题”。

对于一个组件或服务，并发压力增大，响应时间却没有变化，意味着在请求的处理过程中，没有发生对资源的争抢和排队。同理，当响应时间随着并发压力的增大而变长时，一般都意味着这些请求引起了对系统资源的争抢。

对于无状态的服务，理论上可以通过集群扩容，无限增加系统的并发处理能力，简单、直接地解决问题；

但对于有状态的数据服务而言，比如缓存或数据访问，就要考虑资源争抢问题，并针对性地设计、处理。

所以，高性能架构在设计落地时，一个重要的任务，就是去发现那些可能出现资源争抢的模块，并一一进行隔离。

谈到“隔离”，在架构维度有两种方案，一种是在应用层进行隔离，也就是说，在业务功能层面就开始隔离；一种是基础软件层面进行隔离，比如与数据库相关的“读写分离”概念，就是在基础软件层面进行隔离。

对应到具体的实施方法上，有三种主要的“隔离”技巧，你可以多了解：

1. 缓存机制，适用于部分场景，可以解决数据库资源的争抢问题；
2. EDA 架构，适用于处理时间较长的代码逻辑，需要提前区分哪些模块可以做同步处理，哪些模块可以做异步处理；
3. 提前进行预处理，以空间换时间，这也是一种卓有成效的处理手段。

当然了，方法还有很多，听完这节课后，你可以单独再做了解，并将了解到的方法，在评论区分享给大家。

做好隔离后，我们也要注意提升架构的可扩展性，具体方法可以参考 [🔗 《架构设计，专业分工和协作精神的体现》](#)，这里不再重复。

至此，关于高性能设计的三个最重要的步骤就讲完了，我们总结一下，分别是：

1. 为架构做好“保护系统”：做好系统的流量控制；
2. 使架构具备扩容能力：储备额外的计算资源，提升弹性扩容的速度；
3. 提升系统各组件处理能力：识别高并发情境中的资源争抢情况，同时注意保留架构的可扩展性。

最后，我们也要对测试工作保持关注。可能你在 PRE 环境做过压测，但后来发现，实际的业务复杂度是远远超过想象的。

针对这种问题，行业内的解决方案更偏向“全链路生产压测”，说白了，就是在生产环境准备数据、进行压测。如果一个系统通过了全链路压测，那就可以基本确认没有性能问题了。

结语

从我的角度来看，**技术行业发展到今天，基本不存在太多的技术挑战了。**

如果你能将业务问题抽象为一个技术问题，那么不管是寻求同事的帮助，还是 Google、看书、到知识平台付费学习，都能解决你的疑惑。

所以，在理清高性能架构设计的整体思路后，困惑、焦虑，应该从你的认知中消失。具体到数据库设计、缓存设计、队列使用等技术细节，我认为都不是问题。关键之处，是要守住契约，按照我们今天所讲的三大步骤尝试落地。

过去的几年间，我曾在多个企业内建设高性能的 IT 系统架构，很好地服务了用户，使用的就是我们今天所讲的各个方法和思维框架。我个人感觉，过程也是相对轻松的。

需要注意的是，如果时间充足且落实到位，系统确实不会出现高性能问题。可是实际情况是，团队没有时间做完整的高性能设计，没有预算做全链路压测，所以埋下了隐患。

这时，我建议相关负责人要去识别最关键的服务，针对性地进行设计、测试，确保关键服务没有问题，并且，为非关键服务提供降级和熔断处理的开关。

另外一个比较常见的情况，是企业决策层将高性能的设计问题和技术问题混为一谈：看似是技术能力太过欠缺，其实是契约和设计没有做好。**任何复杂问题都可以被拆解为简单问题，只要拆解得足够细致。一定要牢记这一点，这种思维能力是技术人的安身立命之本。**

当然，如果你有问题或想交流的内容，也不要犹豫，立刻在评论区写出来，我会认真回复。

我们下一讲再见。

提建议

训练营推荐

产品经理训练营

成为有大局观、懂用户、能拿结果的产品经理

邱岳 十年资深产品经理，无码科技联合创始人

立抢优惠名额 



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 20 | 高可用设计，让产品没有后顾之忧

下一篇 22 | 扩展性设计，看透业务的本质

精选留言 (10)

 写留言



如水

2020-12-14

技术行业发展到今天，基本不存在太多的技术挑战了。这句话我不赞同，我还没看到中国有好用的操作系统，这个技术没挑战吗？

展开 

作者回复: 你好，如水

谢谢分享、讨论。我们一起讨论：

1 中国操作系统的问题是失去的窗口期，造一个好用的操作系统其实并不难，但难得是周边的生态。所以这个问题目前还真不是技术问题。

2 延申出来看，其实比如登月，人工智能，自动驾驶等等，还是有技术挑战的

3 从第二点来看，可以区分为把这个技术作为一个产品，还是使用产品；比如我们是使用云计算产品，还是研发云计算产品，做一个好的云计算产品还有很大的挑战，但是使用一个云计算产品，没有什么挑战了

4 我们的上下文，中国大部分公司还是commerce公司，就是商业公司，其实更多的是使用技

术，从这点上来说，基本没有挑战，很多的所谓技术挑战只是在重复造轮子。

5 如果觉得自己技术牛，最好的证明就是创业去开一家公司，把自己的技术做成产品卖出去；否则就是在使用技术，那么能不能用好才是衡量的标准

6 但是，放在不同的公司，有可能因为人员能力问题，这些可能还是技术挑战，这是有可能的。我这里说的是趋势，有上下文，就如同做技术的，一定要有context。

谢谢分享，无私地分享你的观点，一起讨论，共同进步，感谢。



4

**Weehua**

2020-12-16

我也非常认同乔老师的观点：在当今中国的互联网环境中，大部分公司都在使用技术，如何使用好技术我认为没有太大的挑战，我们遇到的坑，遇到的技术问题，大部分都是别人走过的路。2年前，带团队开始从0到1做大数据平台相关建设，其中遇到了很多挑战，最后都是通过各种学习和交流得到了很好的解决。

...

展开 ∨

作者回复: 你好，Weehua
握手



2

**李克光**

2020-12-11

"任何复杂问题都可以被拆解为简单问题，只要拆解得足够细致。一定要牢记这一点，这种思维能力是技术人的安身立命之本。"--乔老师很好诠释了《道德经》中“天下难事，必作于易；天下大事，必作于细。是以圣人终不为大，故能成其大”。非常感谢乔老师的技术传道，让我贯通了很多思路，收获良多。

展开 ∨

作者回复: 你好，李克光
谢谢分享，这种领悟和分享特别好，还帮助了别人。



2

**E**

2020-12-11

乔老师看似在讲架构设计，实际上是在讲技术治理，契约精神是治理过程中的指导思想，乔老师也说到契约不可能无上限，因此本质上依然是在给大家布道“Trade-Off”思想。

作者回复: 你好, E

人生永远是在不确定性中寻找确定性, 不断的做决策。



2



行与修

2020-12-13

高可用设计，高性能设计背后是业务设计，架构设计，体系配套设计的全过程方法论。实践中需要以产品为基础持续优化提升，是一个演进过程，无法一蹴而就，往往不是一两个项目所能承载的。回看这三篇，乔老师的课程设计思路也是值得学习的！

展开 ∨

作者回复: 你好, 行与修

功能是basic view，业务、架构都在这个view中，高可用，高性能，监控等是扩展view，所以，都会和basic view有关系。



spark

2020-12-11

对高性能的自顶向下的认知：流量控制、扩容、提升各组件的处理能力

对高性能的分解：

一，架构角度：硬件、异步、中间件、缓存、负载均衡、读写分离，集群，分片；

二，技术角度：网络IO模型、无锁多线程并行、数据结构、算法、网络传输时间、IO时间； ...

展开 ∨

作者回复: 你好, spark

专业，卓越，为你点赞



1



glutton

2020-12-11

还有一个点需要向乔老师请教，目前技术调整或者是技术壁垒基本不存在了，应届生基本对组件啥的很清楚了。。。

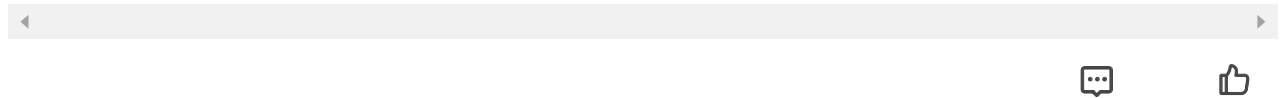
那么，程序员该如何提高自己竞争力呢？听课过程中，有一些点，比如提升认知？改变思维方式？可是感觉又欠缺实现路径。。。

展开 ∨

作者回复: 你好, glutton

T型人才，一定要有自己专业的一个领域，这是一竖； 广度也要有，有专业的领域后，可以向周围延申，不仅仅是技术，还包括软技能，比如沟通能力，决策力，团队管理能力等。

任何事情，认知改变肯定是效果最好的。



glutton

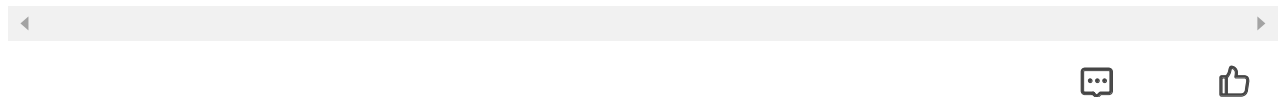
2020-12-11

契约，这个词儿用的很独特，我理解契约其实是目标的另一种表达方式
生产环境压测，才能真实反应系统上限和可能的问题。这块我有切身体会，测试环境压测没有问题，一上生产，耗时不符合预期，最后排查，是某个机房的数据库服务器网卡有问题，虽然这是一个特殊的例子，我是以此来表达，只有生产才是对用户交付的环境。
耗时方面，除了分位数，一些特殊的场景，需要关注下max，虽然压测过程中，tp99数...
展开 ∨

作者回复: 你好, glutton

契约里面首先肯定有目标，还有一层意思是承诺，一诺千金。

关注max，类似tp100了，这种对于时间极度敏感的特别考验技术。



Anyou Liu

2020-12-11

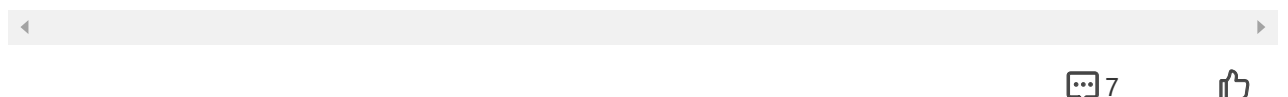
乔老师，测试环境资源有限，配置不如生产，数据量也达不到生产的要求，往往测试环境没问题的，到了生产环境性能还是有问题，测试环境的指标往往也不能直接类比到生产，有没有什么好的解决办法？

作者回复: 你好, Anyou Liu

测试环境肯定资源不能和生产配置一样。

这个时候一种是做好容量规划，测试环境和生产环境和处理有关的参数等比缩小，其他基本参数一样，通过分析进行计算。

第二种是在生产环境进行全链路压测，业界也有这样的工具。



**Geek_2e6a7e**

2020-12-11

加入你提供的服务是数据库，比如TiDB,怎样来提供性能指标。

作者回复: 你好，Geek_2e6a7e

TiDB是无法提供性能指标的，就如同mysql，redis没法提供性能指标。

但一个OMS是有性能指标的，oms的数据库mysql是有性能指标的。通过最外面的服务的性能指标层层进行分解，最后会得出对于oms的mysql数据库的性能指标。

同样的思路可以分析实用了TiDB的应用的性能指标。

